# Peptide Sequence Determination from High-Energy Collision-Induced Dissociation Spectra Using Artificial Neural Networks

Randall E. Scarberry and Zhen Zhang

Department of Biometry and Epidemiology, Medical University of South Carolina, Charleston, South Carolina, USA

Daniel R. Knapp

Department of Pharmacology, Medical University of South Carolina, Charleston, South Carolina, USA

This paper reports a newly developed technique that uses artificial neural networks to aid in the automated interpretation of peptide sequence from high-energy collision-induced dissociation (CID) tandem mass spectra of peptides. Two artificial neural networks classify fragment ions before the commencement of an iterative sequencing algorithm. The first neural network provides an estimation of whether fragment ions belong to 1 of 11 specific categories, whereas the second network attempts to determine to which category each ion belongs. Based upon numerical results from the two networks, the program generates an idealized spectrum that contains only a single ion type. From this simplified spectrum, the program's sequencing module, which incorporates a small rule base of fragmentation knowledge, directly generates sequences in a stepwise fashion through a high-speed iterative process. The results with this prototype algorithm, in which the neural networks were trained on a set of reference spectra, suggest that this method is a viable approach to rapid computer interpretation of peptide CID spectra. *(J Am Soc Mass Spectrom 1995, 6, 947–961)*

The use of high-energy collision-induced dissociation (CID) tandem mass spectrometry for the determination of peptide sequence provides several advantages over the more traditional method of Edman degradation [1]. A major advantage is the very short data generation time, which is typically only a few minutes. Without some automated means to evaluate the data sets collected, however, the time saving is of little consequence because subsequent manual interpretation of the data can require hours to days of an experienced expert's time. The development of reliable computer software for rapid interpretation of CID spectra is, therefore, essential to exploit the time advantage of the mass spectrometric approach to peptide sequencing.

Computer programs to determine peptide sequence from mass spectral data cannot take the traditional computing approach in which all possible sequences are evaluated because, for all but the spectra of very small peptides, there are too many alternatives for the solution to be tractable. For example, greater than $10^{10}$ permutations of the 20 commonly occurring amino acids may account for a nominal peptide mass of 1000 u [2]. Therefore, for a program to analyze a peptide of any appreciable size within an acceptable time period, it must employ some heuristic scheme to significantly reduce the avenues of consideration.

Some early computational approaches limit the possible alternatives nonprogrammatically by requiring additional input from the user before the program begins sequencing. The approach of Hamm et al. [3] requires the identities of all the amino acids present in the sequence. It then generates all possible sequences for evaluation from this reduced set. This approach negates some of the advantages of tandem mass spectrometry, because amino acid analysis must be carried out beforehand to determine the identities of the amino acid constituents. Even without consideration of the added effort and sample consumption entailed, prior knowledge of the amino acid constituents reduces computation time to an acceptable level only if the peptide contains a small number of amino acid residue types.

More recently, programs have been developed that require no initial input other than the tandem mass spectrometry data. Of these, most limit the number of sequences considered by incremental build up of the sequences and periodic purging of all but the most promising sequences from the collection of partial sequences. A program of this category, SEQPEP, devel-

Address reprint requests to Daniel R. Knapp, Department of Pharmacology, Medical University of South Carolina, Charleston, SC 29425-2251.


1044-0305/95/$9.50
SSDI 1044-0305(95)00477-U

Received January 23, 1995
Revised May 12, 1995
Accepted May 14, 1995

oped by Johnson and Biemann [4], is probably the most successful peptide sequencing program to date. With this program, subsequences are constructed one residue at a time beginning from the C-terminus. During an iteration, every subsequence is transformed into 20 child subsequences by the addition of the 20 commonly occurring residues. Following extension, the new subsequences are scored by correlation with ions from the observed mass spectrum. This correlation is achieved by using a rule base to determine which ions should be expected given the proposed subsequence. Because the number of subsequences would otherwise expand explosively in such a scheme, all but the 300 highest-scoring subsequences are eliminated at the conclusion of each iteration.

Even though SEQPEP is a relatively fast program, derivation of sequences depends primarily upon knowledge that the fragmentation process is properly encoded into the knowledge base. Because human understanding of the fragmentation process is presently incomplete, SEQPEP must retain a relatively large number (300) of subsequences after each iteration to prevent the inadvertent elimination of the correct partial sequence during purging, which, nevertheless, sometimes occurs.

The program of Hines et al. [5] takes quite a different approach, in which peaks in the spectrum are classified by using a pattern-based algorithm before generation of sequences from the results. Each of the more intense of the original peaks is hypothetically assigned in turn to each of nine sequence-specific ion categories ($a_n$, $b_n$, $c_n$, $d_{n+1}$, $x_m$, $y_m$, $z_m + 1$, $v_{m+1}$, and $w_{m+1}$) [6] and nine category scores are computed by using a simple function that correlates postulated ion masses with those actually observed. The corresponding $y$ ion mass is computed for each category whose score exceeds a set threshold. These $y$ mass values are accumulated in a table of "$y$ centers" from which sequences are then determined directly. Clearly, the effectiveness of this method hinges upon the ability of the scoring function to correctly classify the original ions. The program attempts to eliminate those peaks that fall into none of the nine classes simply by excluding fragment ions with relative abundances below a constant threshold. A high probability of being in one of these groups, however, may not necessarily follow from a large peak height. Indeed, immonium ions and internal acyl fragment ions are often among the most abundant ions in the spectrum, whereas many of those peaks that belong to one of the nine categories are relatively small.

The sequencing technique presented in this paper incorporates methods similar to both the pattern-based approach of Hines et al. and Johnson and Biemann's iterative technique. As in the program of Hines et al., the ions of the original spectrum are first classified and a new idealized spectrum that consists solely of $y$-type ions is derived from the results. The idealized spectrum is then used by an iterative sequencing scheme

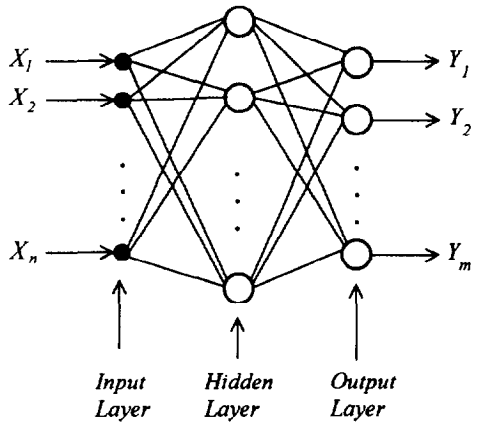
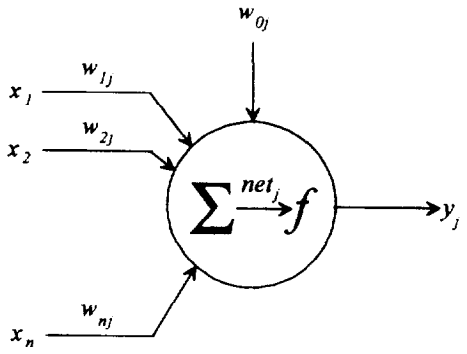



**Figure 1.** An artificial neural network. The neural network receives external values in the form of an $n$-dimensional input vector. The layer of input nodes (dark circles) distribute the inputs to the processing units (light circles) of the first hidden layer. The outputs of the processing units are distributed to the next layer, and so on, until the output layer. This layer emits an $m$-dimensional vector whose elements represent class membership.

that evaluates partial sequences by using much of the fragmentation knowledge described by Biemann et al. [4, 7–10].

The primary new features of this program are the combined use of the two earlier approaches and the use of two artificial neural networks (ANNs) to classify the original ions. Such networks have been used successfully in many varied applications in recent years [11–14]. ANNs are nonlinear computational models for information processing with structures developed based on certain known properties of biological neural systems. They have built-in mechanisms for self-adaptation in response to the data environment. The type of ANN used by this program maps in a nonlinear fashion an $n$-element input vector to an $m$-category classification output vector. The parameters of these two ANNs are determined by exposure to training data to learn the implicit associations between data elements and classification categories. This process does not require the knowledge to be in the form of explicit rules and is, therefore, well suited for pattern recognition tasks in which rules are unknown or are too complicated to be efficiently programmed via a traditional computing approach.

The neural networks used in our program have multilayer perceptron architectures. Such a network is composed of a layer of input nodes followed by one or more layers of processing units (Figure 1). The input nodes simply route the elements of the input vector to each unit of the first layer of processing units through weighted connections. Each processing unit (Figure 2) of the first layer sums its weighted inputs and adds a bias term to compute a net activation signal that is subsequently applied to a nonlinear activation function to compute the unit's output. If another layer exists, the first layer's outputs are similarly routed to its units through more weighted connections. The process con-




$$\textit{Input to unit:} \qquad net_j = w_{0j} + \sum_{i=1}^{n} x_i\, w_{ij}$$

$$\textit{Nonlinear activation function:} \qquad f(x) = \frac{1}{1 + e^{-\theta}}$$

$$\textit{Output of unit:} \qquad y_j = f(net_j)$$

**Figure 2.** A neural network processing unit. The units receives $n$ inputs through weighted links. It adds to the weighted input sum a bias term $w_{0j}$, to compute its net activation $net_j$. This net activation is used as the variable for the sigmoidal activation function to compute the unit output $y_j$.

tinues through the final layer. The outputs of this layer are representations of the network's answers to the classification problem.

Networks with no intermediate layers of processing units between the input nodes and the output layer can only be used to distinguish linearly separable classes [15]. The addition of one or more "hidden" layers, as they have come to be termed, gives a network the ability to perform nonlinear mappings from input space to output space. In fact, a neural network with a single hidden layer may, in theory, implement any functional that maps between two real-valued spaces, provided that the hidden layer contains a sufficient number of units [16, 17]. (However, not all networks may be trainable in a finite time period.) A second hidden layer is sometimes incorporated in a network to enhance training efficiency.

The knowledge of an ANN is entirely contained in the values of its weights and biases. The training of an ANN is a process of adjusting these parameters until the network yields appropriate results by using a set of representative training pattern pairs. A training pattern consists of an input vector paired with a target vector. The elements of the input vector are actual measurements and attributes of the classification subject coded into numerical form, whereas the target vector is the desired response from the network when it is presented with these values. The target vector elements are the representations of the known class membership. The target vector length is, therefore, equal to the number of possible classification categories and each element is set to a value near 1 to signify positive membership or to a value near 0 to signify nonmembership.

The training method used in our program is based on the generalized delta rule described separately by Werbos [18] and Rumelhart and McClelland [19]. It is essentially a gradient descent algorithm to minimize network mean-squared error. The weights and biases of the network are initialized to small random values (typically between $-0.5$ and 0.5), and the network then proceeds through a large number of training sessions or epochs. During an epoch, each of the training pairs is individually presented to the network and, by comparison of the actual network outputs to the target outputs, small weight corrections are computed to minimize the square of the difference (error). If at the end of the epoch, the network mean-squared error has not been reduced to an acceptably low threshold, the weights are adjusted by the cumulative weight corrections and another epoch is begun. When training has been successfully concluded, the weight values are fixed and the network may be used to classify unknown patterns.

## Methods

The program is structured consecutively into three major sections: (1) neural network classification, (2) construction of an idealized spectrum, and (3) derivation of sequences (Figure 3). The algorithm's foundation is the classification of fragment ion peaks by two artificial neural networks in the first module. These networks provide classification measures that the second module uses to construct an idealized spectrum. The third and final module constructs sequences in a stepwise fashion directly from the idealized spectrum by using the existence of idealized peaks to determine which amino acid residues to add in each incrementation step.

### *Neural Network Classification*

A mass spectrum is read from an ASCII file that contains peak masses paired with associated relative abundances. Typically, the spectra used contained masses recorded with precisions of 0.01 u and accuracies of 0.2 u. Such high quality of mass determination makes it possible to store the peak information as an array of data structures indexed by mass. As the spectrum is read, each peak's mass is transformed into an integer through multiplication by the factor 0.9995 and rounding [5]. The representation of masses as integers, in addition to simplification of data storage, facilitates the mass comparisons between peaks, which are performed later in the sequencing module. The constant 0.9995 approximates the reciprocal of the average mass excess for the 20 commonly occurring amino acid residues.


**Figure 3.** The overall structure of the sequencing program that shows the three program modules. The first module computes classification measures for the individual peaks of the original spectrum. The second module uses the results to construct an idealized spectrum that consists of only *y*-type ions. Finally, the third module derives the sequences in a stepwise fashion by using the existence of peaks in the idealized spectrum to direct the extension of incomplete sequences.

Once stored as data structures, the peaks undergo a two phase transformation before they are classified by the neural networks. The first step simply multiplies the peak heights by the constant appropriate to give the height of the protonated molecular ion $MH^+$ a relative abundance of 100.0. In all spectra on which this program was developed, the $MH^+$ ion is the ion of greatest height; therefore, the scaling constant used is simply 100.0 divided by the maximum peak height. The second step is a loop in which the scaled peak heights $h_i$ are transformed into their final values $h_i$, according to the formula

$$h_i = \frac{\ln(h_i + 1.6)}{\ln(100 + 1.6)}$$

The denominator value was selected to force the final peak height of the $MH^+$ ion to be exactly 1.0. Because the $MH^+$ ion has the largest height, the rest of the final ion heights fall into the interval [0.1, 1.0]. This logarithmic transformation helps to compensate for the large height disparities that otherwise would exist between the majority of fragment ions and the prominent $MH^+$ peak and other large ions that often occur at the high mass end of the spectrum. The peak height values are confined to a subinterval of [0.0, 1.0] because peak heights later compose a significant fraction of the neural network inputs. (In general, it is a good practice to constrain the neural network inputs to a small interval to prevent numerical overflow in network computations.) The constant 1.6 in the transformation was selected by trial and error to cause the final peak heights of typical spectra to be well distributed throughout the interval of values. The effect of this second stage of preprocessing on the peak height distributions can be seen clearly in the center graph of Figure 4.

Finally, the two neural networks of the first module are utilized. The purpose of these networks is to compute ion classification measures from which the second module may construct the idealized spectrum. For a given peak $P_j$ the first neural network has a single output element $c_{j0}$ that is indicative of whether or not $P_j$ belongs to one of 11 ion categories. The second network assigns to $P_j$ 11 corresponding class membership scores $c_{ji}$, $i = 1, \ldots, 11$. Both of these classification

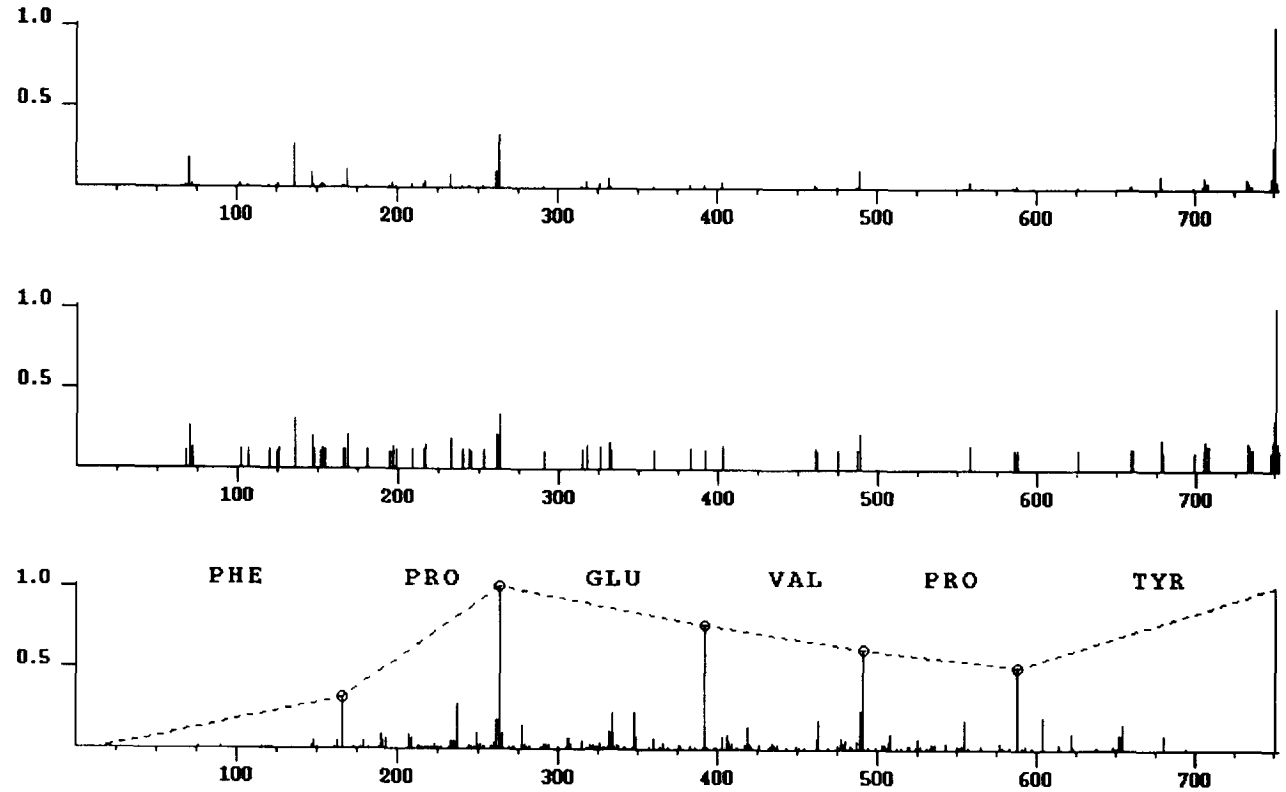



**Figure 4.** Three versions of the spectrum for the peptide YPVEPF. On top is a plot of the original spectrum (magnified by 10). The center shows the same spectrum after logarithmic transformation. The idealized spectrum is shown at bottom with a dotted trace for the true sequence that connects prominent *y* ions.

networks operate on the same input vector format. The elements of the input vector are values that are assumed to be pertinent to determine whether the ion under consideration is one of the classification categories and, if so, to which of these categories the ion belongs. (See further discussion that follows.)

The classification categories referred to are the six major ion types ($a_n$, $b_n$, $c_n$, $x_m$, $y_m$, and $z_m + 1$) that result from cleavages along the peptide backbone and the ions $d_{n-1}$, $v_{m-1}$, and $w_{m-1}$ that result from side chain losses. (There are 11 categories rather than 9, because $d_{n-1}$ and $w_{m-1}$ ions are computed by using beta substituents equal to both H and $CH_3$.) A family that consists of all of these ion types can be associated with a specific cleavage location along the peptide backbone (Figure 5). All the ions in the family are centered about the cleavage location that gives rise to the $y_m$ and $b_n$ ions. From the ion structures illustrated in Figure 5, one may easily determine the mass relationship between ions in a family. For example, the $x_m$ ion differs from the $y_m$ ion in that it possesses an additional carbon and oxygen atom, but does not retain the two protons. Its mass is, therefore, the mass of the $y_m$ ion plus 26. The second column of Table 1 gives the simple formulas that relate ion family masses to the $y_m$ ion mass. The subscripts $n$ and $m$ are used in the same ion family because N-terminal ions $a_n$, $b_n$, $c_n$, and $d_{n-1}$ will, in general, have a different subscript from the C-terminal ions of the same family. The two subscripts will always add to the total number of amino acid residues that compose the peptide.

If an ion at a given mass location were postulated to belong to a given classification, the other ions in the same family could be computed by using the formulas of Table 1. However, if the ion were assigned to a different classification, an entirely different set of ion family masses would result. For example, for a peptide with an $MH^+$ mass of 843 u, if an ion with a nominal mass of 350 u is assumed to be an ion of type $a_n$, the ion family shown in the top of Figure 6 results, whereas the ion family in the bottom of Figure 6 results if the ion is assumed instead to be of type $x_m$. Thus, by hypothetical assignment of an ion to each of the 11 ion types, 11 distinct hypothetical ion families can be derived. (Note that 11 types result by considering $R_n$ = H and $CH_3$ for $d$ and $w$ ions.)

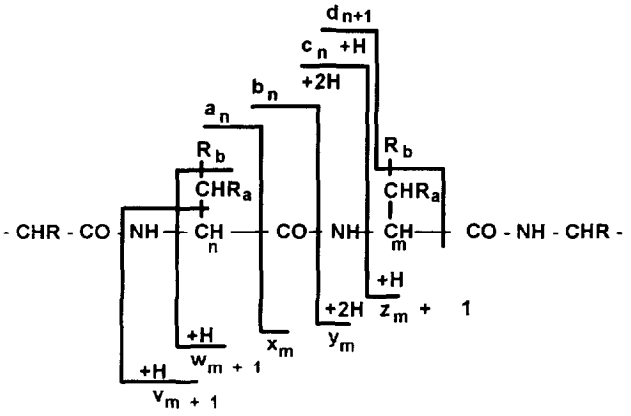



**Figure 5.** The nine types of fragmentation ions that compose an ion family.

If one were then to attempt to manually classify the ion, a likely method would be to correlate each of these hypothetical ion families to ions actually present in the spectrum. One might conclude that the ion belongs either to the hypothetical class that best correlates with the actual spectrum or to none of the classes if all correlate poorly. This reasoning was used to structure the input vector format for the two neural networks. The majority of the input vector elements are the actual peak heights at the mass locations that correspond to the hypothetical ion family masses. The networks may, therefore, "observe" the hypothetical class correlations themselves from the actual data.

**Table 1.** Ions that constitute a cleavage family[a]

| Ion Type | Ion mass | Ion structure |
|---|---|---|
| Parent | $MH^+$ | $[H-(NH-CHR-CO)_{n-1}-NH-CHR_n-CO-NH-CHR_m-CO-(NH-CHR-CO)_{m-1}-OH]H^+$ |
| $a_n$ | $MH^+ - m_y + 27$ | $H-(NH-CHR-CO)_{n-1}-NH^+=CHR_n$ |
| $b_n$ | $MH^+ - m_y + 1$ | $H-(NH-CHR-CO)_{n-1}-NH-CHR_n-CO$ |
| $c_n$ | $MH^+ - m_y - 18$ | $[H-(NH-CHR-CO)_n-NH_2]H^+$ |
| $d_{n-1}$ | $MH^+ - m_y + 43 - R_n$ | $[H-(NH-CHR-CO)_n-NH-CH]H^+$ (CHR$_n$ double-bonded) |
| $v_{m-1}$ | $m_y + 55$ | $NH=CH-CO-(NH-CHR-CO)_m-OH]H$ (CHR$_1$ double-bonded) |
| $w_{m-1}$ | $m_y + 53 - R_n$ | $[CH-CO-(NH-CHR-CO)_m-OH]H^+$ (CH double-bonded) |
| $x_m$ | $m_y + 26$ | $CO-(NH-CHR-CO)_m-OH$ |
| $y_m$ | $m_y$ | $[H-(NH-CHR-CO)_m-OH]H^+$ |
| $z_m + 1$ | $m_y - 16$ | $[\cdot CHR_m-CO-(NH-CHR-CO)_{m-1}-OH]H^+$ |

[a] $MH^+$ represents the total peptide mass; $R_n$ is the beta substituent mass. The subscripts $n$ and $m$ add to the sequence length of the peptide.

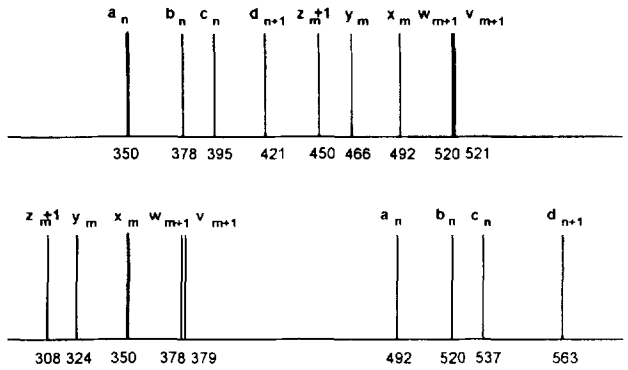



**Figure 6.** Two ion family interpretations for an ion with mass 350 u from a peptide with $MH^+$ that has 876 u. The top shows the ion family that results if the ion is considered to be an $a_n$ type. The entirely different family of the bottom diagram results when the ion is assumed to be in the $x_m$ category. (For simplicity, only the $d_n$ and $w_m$ ions that use $R_i$ = H are shown in both diagrams.)

Also deemed pertinent to a peak's classification is its proximity to either end of the spectrum. Non-sequence-specific fragments tend to be located more prominently in these regions. Therefore, the input vector also contains two measures indicative of the closeness of the peak to the low and high mass ends of the spectrum. Each of these measures is scaled linearly to correspond to a 0–200-u range. For example, if the ion possesses a mass of 100 u, the distance measure that indicates the proximity to the low mass end will be 0.5; if the ion mass is 200 u or larger, the distance measure will be 1.0. Again, because these measures are neural network inputs, the proximity measures are constrained to the interval [0.0, 1.0]. The upper bound of 200 u was introduced because it was reasoned that for ions located more than 200 u from the endpoint, the exact proximity bears little relevance to classification.

The rest of the input vector elements are statistical measures that characterize the overall spectrum because it was thought that the nature of the spectrum might have some bearing on fragment ion classification. Two such measures are the mean peak height and the peak height standard deviation. Also included are the peak mean mass, the total number of peaks divided by the length of the spectrum (peak density), and a "center of height" $m_h$ given by

$$m_h = \frac{\sum m_i h_i}{\sum h_i}$$

where $m_i$ and $h_i$ represent the mass and peak height of the $i$th fragment ion. This term is similar to the center of mass of a set of discrete particles arranged in a straight line. However, in this case, the locations are specified by the peak masses whereas the heights are the weighting factors analogous to the particle masses. For their values to be in the interval [0.0, 1.0], the mean peak mass and $m_h$ are both divided by the length of the spectrum, $MH^+$.

Finally, the input vectors contain 20 elements representative of 2 histograms each with 10 divisions. The first is a histogram of the peak height distribution; the other is a histogram of peak masses multiplied by their associated heights. The latter histogram provides the networks with an overall shape profile of the spectrum. The transformed spectrum of the tripeptide MRF and its shape histogram are shown in Figure 7. Both of the histograms are normalized so that the sum of their elements is 1.0.

The classification itself is performed in a single pass through the array of spectral peaks. For each peak, other than the $MH^+$ ion, the input vector is constructed and presented to each of the neural networks. The single output of the first network along with the 11 output elements of the second network are stored as part of the associated peak data structures in the array.

## *Construction of Idealized Spectrum*

The neural network classification results are the raw material used to construct the idealized spectrum. The idealized spectrum is composed only of $y$-type ions, but any of the other ion types just as easily could have been used because sequences are derived by observation of mass differences between adjacent ions of the same type. The idealized spectrum is represented in a separate array of data structures with a length equal to that of the original spectrum. The new spectrum is built by once again performing a loop through the peaks of the actual spectrum to compute contributions for the initially empty idealized array. This process is described in the following lines of pseudocode:

```
For each peak, P_i, do:
  For each classification category, i, do:
    Calculate mass, m_ji, of associated y ion
    If m_ji is valid, then augment height in idealized
      spectrum at index m_ji by (C_j0 * C_ji)
  endloop;
endloop;
```

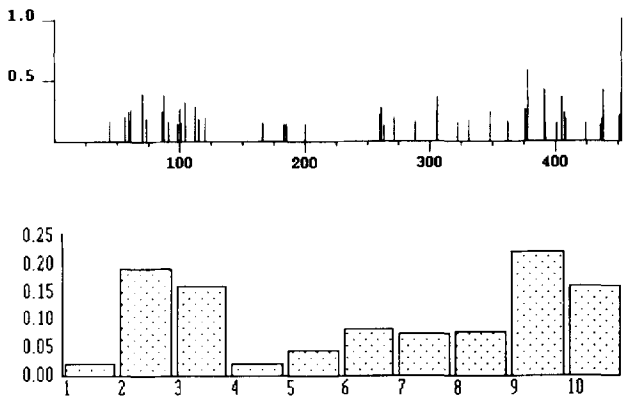



**Figure 7.** Transformed spectrum (top) and shape histogram (bottom) for the tripeptide MRF. The elements of the shape histogram are used as neural network inputs to convey the general height distribution of the spectrum.

In the inner loop, $c_{j0}$ and $c_{j1}$ represent the results of the neural networks. $c_{j0}$ is the output of the first network for peak $P_j$, and $c_{j1}$ through $c_{j11}$ are the outputs given by the second network.

Within the inner loop, a validation check is used to reduce the overall number of idealized peaks. A peak at mass $m_{ji}$ is not incremented if it is impossible for a $y$ peak to exist with that mass. A mass of 60 u would not be incremented, for example, because there is no combination of amino acid residues that can yield such a mass. This check is performed quickly by using a static lookup table constructed by another program. (It should be noted that this initial implementation of the program is designed to deal only with unmodified peptides; future extensions of the program will require expansion of the lookup table.)

Each original peak thus contributes to as many as 11 peaks in the idealized spectrum, that is, the $y$ peaks that correspond to the original peak are of any of the 11 peak types. Ideally, however, for a peak that belongs to one of the fragmentation categories, one contribution will be significantly larger than all the rest. If the neural networks have performed the peak's classification well, the peak's contribution to a given idealized peak is proportional to two factors: (1) the likelihood that the peak $P_j$ is not an extraneous peak as expressed by $c_{j0}$ and (2) the likelihood that the peak is from the class used to derive the mass $m_{ji}$ of the idealized peak as expressed by $c_{ji}$. (Strictly speaking, the neural network outputs are not probability measures; because they are confined to the interval [0.0, 1.0], however, it is convenient to view them as such.) Even though the idealized spectrum will contain many peaks, the cumulative effect from networks with reasonably accurate classification results is the predominance of a relative few. As a final step, the idealized spectrum is scaled to the interval [0.0, 1.0]. The idealized spectrum of the peptide, YPVEPF, is shown in the lower graph of Figure 4 with a dotted line tracing over the true sequence.

## *Sequence Derivation*

The sequencing module, which is diagramed in Figure 8, is the most programmatically complex of the three major sections. Both complete and partial sequences are stored in a variable length list, which is initialized with a single empty or null sequence. The algorithm then performs a loop in which sequences are incrementally built up and exit when, at the end of an iteration, all sequences in the sequence list account for the mass of the peptide. Among the 20 commonly occurring amino acids, there are two pairs of residues with the same nominal masses: the isomers isoleucine and leucine, and glutamine and lysine, which are isobaric. Within the sequencing loop, leucine and isoleucine are not distinguished separately; they are represented by a single extension placeholder Lxx. Glutamine and lysine are, however, treated separately because of differences in the manner in which they influence fragmentation. More details that concern the way this pair of amino acids is handled are given subsequently. When all sequences in the list are complete, the program exits from the sequencing loop and attempts to differentiate between leucine and isoleucine before the final results are displayed.

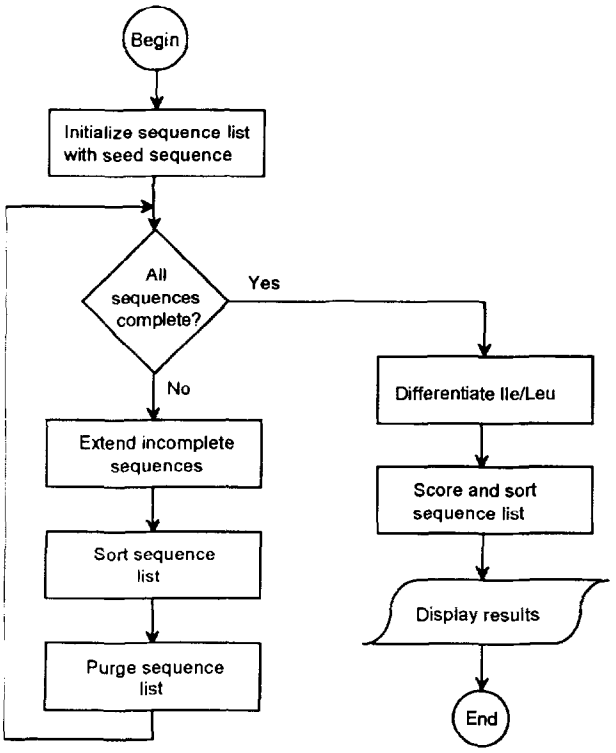



**Figure 8.** Flow diagram of the sequence derivation module. The seed sequence that contains no amino acid residues is provided to start the first iteration of the sequencing loop. The incomplete sequences are extended by each amino acid for which there is a supporting $y$-type ion in the idealized spectrum. The extended sequences are assigned scores that allow the list to be sorted in descending order. At the conclusion of each iteration all but the top 50 sequences are purged. The loop is exited when all sequences in the list have masses that match the mass of the peptide. The program then differentiates between isoleucine and leucine, resorts the list, and displays the sequence list.

Sequence extension is done a single amino acid residue at a time from the N-terminus. Each sequencing iteration first extends the incomplete sequences in the sequence list. An incomplete sequence is removed from the sequence list and transformed into 19 child sequences by the addition of each of the 19 amino acid residue masses on its C-terminus. A given child sequence is discarded unless it satisfies one of two criteria: (1) it is a complete sequence or (2) the latest amino acid extension is supported by the existence of an idealized $y$ peak. The child sequence is complete—accounts for the overall mass of the peptide—if its mass plus 19 (the combined mass of the C- and N-terminal groups) is equal to $MH^+$. If it is not complete, the mass for the idealized $y$ peak that results from the latest extension is computed easily by subtracting the

child sequence's mass from MH$^+$. If, in the idealized spectrum, a $y$ peak exists at this mass, the sequence is added to the sequence list.

Because the sequence list is periodically purged of all but the most promising sequences, newly extended sequences must be given some measure of their worth. This is achieved by assignment of a score to each newly extended sequence as it is added to the sequence list. A sequence's score S is a linear combination of three variables given by

$$S = c_1 r_a + c_2 r_p + c_3 r_i$$

where $c_1$, $c_2$, and $c_3$ are constants. The simplicity of this equation is deceptive, because a great deal of complexity is hidden in the computations of the three variables. These variables are (1) the idealized peak height ratio $r_a$, (2) the original peak height ratio $r_p$, and (3) the peak presence ratio $r_i$.

The idealized peak height ratio is a measure of how well the sequence is corroborated by information in the idealized spectrum. Because the idealized spectrum is built solely from the outputs of the neural networks, this variable, in essence, expresses the degree to which the neural networks agree with the sequence interpretation. The formula for $r_a$ is

$$r_a = \frac{y_{in}}{y_{out} + 0.1}$$

The computation of $r_a$ takes into account the idealized $y$ peaks that match the expected $y$ peaks, given the proposed sequence, and it also takes into account those that do not match. $y_{in}$ is the average height of peaks that correlate with the sequence, and $y_{out}$ is the average height of the peaks that the sequence passes over. For example, $y_{in}$ for the sequence shown in the lower graph of Figure 4 is the average height of the five idealized peaks connected by the sequence trace. $y_{out}$ is the average height of the remaining peaks. (The reason that 0.1 is added to the denominator is to prevent division by zero errors on the occasions when $y_{out}$ equals zero.)

The far more intricate computations of $r_p$ and $r_i$ are performed in tandem. Both of these are measures of how well the original spectrum correlates with the proposed sequence. $r_p$ is the average height of peaks actually present in the original spectrum that are expected to exist given the sequence. The term $r_i$ is the fraction of these expected peaks that are present. In these computations, an expected peak is not considered to be present unless its height exceeds a set threshold.

The difficulty in computing $r_p$ and $r_i$ stems from determination of which peaks to expect. To accomplish this task, the program employs a rule base that embodies much of the fragmentation knowledge described in the literature. When a sequence is scored, an array of flag variables—the expected ion array—is used to record whether a peak of a given category is to be expected at each mass location. For example, if in accordance with the rule base a $b_n$ ion is to be expected at mass location 124 u in the original spectrum, then a $b$ ion flag will be set at index 124 in this array. (Because of the programming techniques used, more than one category of peak can be flagged for a given mass location.) In each sequence scoring, all set flags are initially flushed from the array. The scoring procedure iterates through each of the proposed sequence's cleavage sites and sets the ion flags as appropriate. After all of the flags are set, it is a simple matter for the program to loop through each mass location in the expected ion array to check the original spectrum for a peak whenever it encounters a flag.

The sequence-specific ions for which the program checks to score a sequence are the categories $a_n$, $b_n$, $c_n$, $d_n$, $v_m$, $w_m$, $x_m$, $y_m$, $y_m - 2$, $z_m$, and $z_m + 1$. For a given cleavage along the peptide backbone, the ions to be flagged in the expected ion array are determined by using the following rules:

1. Ions of type $a_n$ are not flagged when the cleavage site is N-terminal to a basic amino acid (arginine, histidine, lysine) unless there is another basic amino acid located further toward the N-terminus of the sequence [4]. Also, $a_n$ ions are not flagged when the cleavage site is C-terminal to glycine [9].
2. An ion of type $c_n$ is flagged only if the associated $a_n$ and $b_n$ ion are both flagged and actually present in the spectrum [4] and if the amino acid C-terminal to the cleavage is threonine, tryptophan, lysine, or serine [10].
3. An ion of the $d_n$ category is flagged only if the associated $a_n$ ion is flagged and observed in the spectrum. In addition, the amino acid on the side N-terminal to the cleavage cannot have a side chain with less than two carbon atoms (glycine and alanine), is not aromatic (phenylalanine, tyrosine, histidine, tryptophan), and is not cyclic (proline) [4, 7].
4. The only cleavage sites for which $y_m$ ions are not flagged are those C-terminal to proline [9].
5. The $y_m$-2 ion is only flagged for cleavages N-terminal to proline [9].
6. The $z_m$ and $z_m + 1$ ions are flagged unless the cleavage site is N-terminal to proline [4].
7. For ions of type $v_m$ to be flagged, a basic amino acid must be located in the sequence C-terminal to the point of cleavage. Also, the C-terminal amino acid extension must be an aromatic amino acid (phenylalanine, tyrosine, histidine, tryptophan), aspartic acid, or an amino acid with a $\beta$ substituent (serine, valine, isoleucine, threonine) [4, 7].
8. $w_m$ ions are flagged if a basic amino acid is located C-terminal to the point of cleavage and if the C-terminal amino acid is not aromatic (phenylalanine, tyrosine, histidine, tryptophan) and is not an amino acid with less than two carbon atoms in the side chain (glycine, alanine) [4, 8].
9. Ions in the categories $b_n$ and $x_m$ are flagged for every cleavage site.

For an incomplete sequence, the rules that require knowledge of the amino acid on the C-terminal side of the cleavage cannot be completely implemented for the cleavage site associated with the final amino acid extension. In general, the ions whose occurrences depend heavily on the presence of particular amino acids C-terminal to the cleavage are not flagged for this site ($c_n$, $y_{m-2}$, $v_m$, and $w_m$). Ions of the types $a_n$, $z_m$, and $z_m + 1$, however, are flagged if they satisfy the other rule stipulations.

Unlike the $d_n$ and $w_n$ computations in construction of the neural network input vectors, which use two beta substituent ($R_\beta$) values of H and $CH_3$, the scoring procedure uses the $R_\beta$ substituents appropriate for the amino acids adjacent to the cleavage site. These are OH for threonine, $C_2H_5$ for isoleucine, $CH_3$ for valine, threonine, and isoleucine, and H for the amino acids not branched at the beta carbon. The reason for this difference is that sequences are scored with a prior knowledge of the amino acids involved in the proposed sequence. The input vectors are constructed with no knowledge of the amino acids so, not to enlarge the input vector inordinately, only the two most common values for $R_\beta$ are used.

In addition to the sequence-specific ion types already described, the scoring process also flags the expected ion array for the presence of some other ion categories. These are the amino acid immonium ions—ions that result from the loss of single amino acid side chains—and internal acyl fragment ions. The first two of these are especially helpful as indications of the presence of a particular amino acid in the peptide [1].

An additional explanation must be given with regard to the manner in which sequences are extended by glutamine and lysine. The fact that lysine is a basic amino acid results in its presence having a dramatic effect upon which part of the cleaved peptide will retain the positive charge. Its presence C-terminal to the cleavage results in an increased likelihood that the C-terminal cleavage ions ($y_m$, $x_m$, $z_m$) will result or vice versa if it is located N-terminal to the cleavage. Both glutamine and lysine are, therefore, included in sequence extension, but, of the two new sequences, the one with the lower score is immediately discarded to reduce the population of the sequence list. In earlier versions of the program in which both sequences were retained, it was observed that the two sequences usually, with occasional noteworthy exceptions, had almost identical scores.

When all the incomplete sequences have been replaced by newly extended and scored sequences, all but those with the 50 highest scores are deleted. If incomplete sequences are still present among the remainder, the sequence extension loop repeats; otherwise, the algorithm breaks out of the loop and proceeds with the differentiation of leucine and isoleucine.

Because leucine and isoleucine have different beta substituents, the placement of their observed $w_m$ and $d_n$ ions may be used to distinguish between them. Leucine's single beta substituent is a hydrogen atom whereas isoleucine has both $CH_3$ and H. Because both the $w_m$- and $d_n$-type ions involve the loss of a single substituent, the masses at which these ions are found can be computed by using the equations of Table 1. For each sequence amino acid labeled as Lxx in the sequencing loop, the differentiation step computes two measures. The first, $h_{\mathrm{Leu}}$, is the average height of the two peaks in the original spectrum found at the masses for the $d_n$ and $w_m$ ions computed by using H for $R_\alpha$. Similarly, the second, $h_{\mathrm{Ile}}$, is the average height for the four peaks computed by using both $CH_3$ and $C_2H_5$ for $R_\alpha$. Then the ratio of these two measures is compared against a constant threshold $T_{\mathrm{Leu/Ile}}$ as follows to arrive at a classification decision:

$$\text{If} \quad \frac{h_{\mathrm{Leu}}}{h_{\mathrm{Ile}}} > T_{\mathrm{Leu/Ile}}, \qquad \text{classify as leucine.}$$

$$\text{If} \quad \frac{h_{\mathrm{Leu}}}{h_{\mathrm{Ile}}} < (T_{\mathrm{Leu/Ile}})^{-1}, \qquad \text{classify as isoleucine.}$$

If

$$(T_{\mathrm{Leu/Ile}})^{-1} \leq \frac{h_{\mathrm{Leu}}}{h_{\mathrm{Ile}}} \leq T_{\mathrm{Leu/Ile}},$$

classify as indeterminate.

## Results and Discussion

A collection of 43 preclassified reference spectra was used in the development and testing of the prototype program. All but one of the spectra were generated on the JEOL HX110/HX110 tandem mass spectrometer (JEOL, Peabody, MA) in our laboratory. [The spectrum for DVVLVDAGLK, generated on a Kratos Concept II instrument (Kratos Analytical, Ramsey, NJ) was obtained from the University of California, San Francisco Mass Spectrometry Resource.]

The object of experimentation was to classify each of the reference spectra multiple times by using unbiased neural networks, that is, networks that actually had not been trained on data from the spectra under examination. These aims were accomplished by performing three independent testing runs, during each of which all of the spectra were classified once. To prepare for a given testing run the list of spectra was ordered randomly and then divided into four approximately equal partitions (11, 11, 11, and 10 spectra). Networks to test the spectra of a given partition were trained on training vector pairs constructed from data in the other three partitions of the given run, that is, the spectra in partition A were tested by using networks trained on the data from partitions B, C, and D pooled as a single training set; those in partition B were tested by using networks trained on the data from partitions A, C, and D, and so forth. These four tests constituted a "run." Because three independent runs of four partitions were performed, it was neces-

sary to train 12 independent neural network pairs on 12 separate sets of training data.

The data sets for training these neural network pairs were derived by using a separate C language program on a Zeos International computer with an Intel 80486 CPU. Given the known peptide sequence, the program created two training data files for each spectrum—one for each of the two network architectures. These data files, written in ASCII format, were then ported to a UNIX workstation (DEC, Nashua, NH) where they were combined as appropriate to construct the training data set for each set of neural networks.

As a result of testing the training performance of a number of network architectures, both networks in each test set were given a single hidden layer of 60 processing units and trained for 500 epochs. Networks of the first type achieved approximately 85% accuracy on the training sets; networks of the second type achieved approximately 90% accuracy. Each of the trained network pairs was separately incorporated into the program to sequence the spectra of the associated partition.

The results of this testing on the 43 spectra are shown in Table 2. The first column displays the known sequence. The second column lists the partitions, designated A, B, C, or D, to which the peptide spectra belonged during the three respective testing runs. (For example, peptide AA was in partition D in the first run, partition C in the second run, and partition A in the third run.) The third and fourth columns give the orders in which the correct interpretations were found in the program's output list for each of the three runs. The entire testing process was performed once with fixed scoring constants to obtain the results of column three. Column four lists the results obtained when the tests were repeated on the same neural networks but separately derived scoring constants for each partition. The final column lists the average time in seconds taken to generate the sequences.

During the evolution of the program, scoring constants $c_1$, $c_2$, and $c_3$ equal to 2.0, 0.55, and 2.0 were manually selected through a trial-and-error process to produce optimal results. These are the constants used to obtain the ranks in column three. However, this column cannot be regarded as truly indicative of how the program would perform on unknown spectra in general, because the constants were not derived independently of the test spectra. These results were included, nevertheless, to demonstrate the stability of interpretation between runs when the neural networks were the only items that were varied.

To obtain the more objective results of column four, scoring constants were derived separately for each partition via a technique that optimized the sequencing results when the program was run on the spectra in the other three partitions of the associated run. In this way, neither the neural networks nor the scoring constants used to test a partition had "seen" the spectra in the partition.

The scoring function $S$ can be thought of as the inner product of two vectors, $\mathbf{C} = [c_1, c_2, c_3]^T$ and $\mathbf{r} = [r_y, r_h, r_p]^T$. If we have the vector $\mathbf{r}$ of the true peptide sequence and the vector of an incorrect sequence, designated $\mathbf{r}^*$, then if the scoring function operates properly, $S > S^*$ ($S^* = \mathbf{C}^T\mathbf{r}^*$). Thus, if we define $\mathbf{X} = (\mathbf{r} - \mathbf{r}^*)$, we should have $\mathbf{C}^T\mathbf{X} > 0$.

The trained neural networks to be used to test a given partition were incorporated into the program, the scoring ratios were weighted equally ($\mathbf{C} = [1.0, 1.0, 1.0]^T$), and the program was used to sequence each of the spectra in the other three partitions of the run. For each spectrum, if the true sequence was found in the final sequence list, its vector $\mathbf{r}$ and the vectors $\mathbf{r}^*$ of the top 10 incorrect sequences were used to construct 10 $\mathbf{X}$ vectors. These vectors were accumulated into a data file of vectors $\mathbf{X}_i$, $i = 1, \ldots, K$, on which the linear learning machine [20] optimization technique was employed to derive final values of $\mathbf{C}$ for testing the partition as follows:

1. Set $\mathbf{C} = [3^{-0.5}, 3^{-0.5}, 3^{-0.5}]^T$.
2. Set $\Delta\mathbf{C}_n = 0$.
3. For each vector $\mathbf{X}_i$, $i = 1, \ldots, K$: If $\mathbf{C}_n^T\mathbf{X}_i < 0$, augment $\Delta\mathbf{C}_n$ by $\gamma\mathbf{X}_i$, where $\gamma = -0.001\mathbf{C}_n^T\mathbf{X}_i/\mathbf{X}_i^T\mathbf{X}_i$.
4. Set $\mathbf{C}_{n+1} = \mathbf{C}_n + \Delta\mathbf{C}_n$.
5. Normalize $\mathbf{C}_{n+1}$.
6. Loop back to step 2 until $\mathbf{C}$ stabilizes.

$\mathbf{C}$ was normalized throughout the algorithm to prevent its elements from taking on extremely large values. Approximately 2500 iterations of the loop were required during each derivation for the constants to arrive at stable values. The values of the constants for each partition are displayed in Table 3.

It is apparent from examination of column three of Table 2 that the program gave reasonably consistent performance over the three runs when only the neural networks were varied, because 34 (79%) of the sequences received the same rank over the three tests. Twenty-seven (63%) of the correct sequence interpretations received the top rank in the sequence list in all three test runs, whereas 40 (93%) placed in the top five.

In only two cases:

NDIAAK and HGTVVLTALGGILK

were the correct sequences not found in the sequence list. The reason that NDIAAK was not listed is that the LEU/ILE differentiation step misclassifies the isoleucine residue as leucine. In all three testing runs, however, NDLAAK is listed as the top sequence. In the case of HGTVVLTALGGILK, the sequence HGTVVLTAXZVLZ is the top scoring sequence in every run. The spectrum of HGTVVLTALGGILK was closely examined and it was found that no ions in the family that resulted from the cleavage between the two glycines were present. Thus, there was no

**Table 2.** Results of sequencing program on preclassified reference spectra in the three independent testing runs[a]

| Peptide | Test partitions | Rank (constants fixed) | Rank (constants variable) | Avg CPU time (s) |
|---|---|---|---|---|
| AA | D, C, A | 1, 1, 1 | 1, 1, 1 | 0 |
| AAA | D, A, C | 1, 1, 1 | 1, 1, 1 | 0 |
| AAAAA | B, C, A | 1, 1, 1 | 1, 1, 1 | 0 |
| AKTE | D, D, A | 2, 2, 2 | 2, 2, 6 | 0 |
| ALELFR | C, B, C | 2, 2, 2 | 1, 1, 1 | 2 |
| ASTTVSKTE | D, C, B | 1, 1, 1 | 1, 1, 2 | 4 |
| DDE | C, D, D | 1, 1, 1 | 1, 1, 1 | 0 |
| DVVLVDAGLK | C, B, B | 1, 2, 5 | 7, 7, 7 | 4 |
| EDLIAY | B, B, D | 2, 2, 2 | 4, 3, 2 | 1 |
| EDLIAYLK | A, B, B | 4, 3, 3 | 2, 3, 1 | 4 |
| EMPFPK | B, A, D | 1, 1, 1 | 1, 1, 1 | 2 |
| ETTTDK | A, B, C | 1, 1, 1 | 1, 1, 1 | 2 |
| FVQWLMNT | B, A, B | 1, 1, 1 | 1, 1, 1 | 3 |
| GGGG | A, D, C | 1, 1, 1 | 1, 1, 1 | 1 |
| GITWK | A, C, A | 1, 1, 1 | 1, 1, 1 | 1 |
| GLLG | B, A, B | 2, 2, 2 | 3, 3, 2 | 1 |
| HGTVVLTALGGILK | A, D, B | – | — | 6 |
| HKIPIK | D, C, D | 1, 2, 1 | 1, 4, 2 | 2 |
| IFVQK | A, B, B | 1, 1, 1 | 1, 1, 1 | 1 |
| IHPF | B, B, B | 1, 1, 1 | 1, 1, 1 | 1 |
| III | B, A, D | 1, 1, 1 | 1, 1, 1 | 1 |
| IVV | D, D, B | 1, 1, 1 | 1, 1, 1 | 0 |
| LFTGHPETLEK | C, D, C | 3, 2, 4 | 7, 3, 6 | 7 |
| LGG | D, D, A | 1, 1, 1 | 1, 1, 1 | 0 |
| LLVY | A, B, D | 1, 1, 1 | 1, 1, 1 | 0 |
| LRRASLG | D, A, C | 3, 3, 3 | 4, –, 17 | 2 |
| MAS | C, D, A | 1, 1, 1 | 1, 1, 1 | 1 |
| MGMM | A, A, D | 1, 1, 1 | 1, 1, 1 | 1 |
| MIFAGIK | B, B, D | 1, 1, 1 | 1, 1, 1 | 3 |
| MRF | A, A, C | 1, 1, 1 | 1, 1, 1 | 1 |
| NDIAAK | C, B, A | — | — | 2 |
| QEPVLGPVR | A, C, D | 2, 3, 3 | 2, 2, 2 | 3 |
| SGAGAG | D, D, A | 1, 1, 1 | 1, 1, 1 | 1 |
| TGPNLHGLFGR | C, C, C | –, 15, 17 | 6, 4, 5 | 8 |
| TKY | B, C, C | 2, 3, 4 | 2, 2, 3 | 0 |
| TSQVAPA | C, A, B | 1, 1, 3 | — | 2 |
| TYSK | C, C, A | 1, 1, 1 | 1, 1, 1 | 1 |
| VLPVPQK | B, B, B | 1, 1, 1 | 1, 1, 1 | 2 |
| VLS | A, A, D | 1, 1, 1 | 1, 1, 1 | 0 |
| VLSEG | B, C, C | 1, 1, 1 | 2, 2, 2 | 1 |
| YIPGTK | C, A, A | 1, 1, 1 | 2, 2, 2 | 2 |
| YKT | C, D, C | 1, 1, 2 | 1, 1, 1 | 1 |
| YPVEPF | D, C, A | 1, 1, 1 | 1, 1, 1 | 1 |

[a] Column two indicates the partition of the peptide for the three testing runs. Column three displays the test results when fixed scoring constants (2.0, 0.55, 2.0) were used. Column four shows the results when the constants shown in Table 3 were used. Column five is the average time taken to sequence the peptide. (Absence of rank figures indicates the correct sequence was not among the 50 highest scoring sequences. An indicated CPU time of 0 s indicates less than 0.5 s.)

**Table 3.** Scoring constants derived for each partition of the three testing runs (the sequencing results when these constants were used are displayed in column four of Table 2)

| Test partition | $C1$ | $C2$ | $C3$ |
|---|---|---|---|
| 1A | 0.396 | 0.413 | 0.82 |
| 1B | 0.409 | 0.478 | 0.777 |
| 1C | 0.364 | 0.46 | 0.81 |
| 1D | 0.55 | 0.276 | 0.788 |
| 2A | 0.394 | 0.474 | 0.787 |
| 2B | 0.343 | 0.477 | 0.809 |
| 2C | 0.437 | 0.447 | 0.781 |
| 2D | 0.673 | 0.33 | 0.662 |
| 3A | 0.542 | 0.383 | 0.748 |
| 3B | 0.342 | 0.407 | 0.847 |
| 3C | 0.398 | 0.444 | 0.803 |
| 3D | 0.405 | 0.394 | 0.825 |
| Average values | 0.438 | 0.415 | 0.788 |

resulting $y$ peak for this cleavage in the idealized spectrum and, consequently, the incomplete sequence HGTVVLTAL was never extended by glycine.

Column four displays somewhat surprising results as performance actually improved on six of the spectra, most notably on that of TGPNLHGLFGR. Twenty-six (60%) of the correct sequences received the top rank, whereas 35 (81%) placed in the top five. Performance declined on 11 of the spectra, particularly on those of LRRASLG and TSQVAPA. To determine the cause of the decreased performance on TSQVAPA, the sequencing list was inspected at the end of each extension step. It was learned that, with the new constants, the incomplete sequence TSQ was purged from the list after the third extension. When the correct sequence is eliminated at any stage of sequencing, the true sequence will not appear in the final list. The program was modified to allow manual insertion of sequences into the final sequence list to see how TSQVAPA would fare. When it was inserted, it placed first in all three runs.

Because TSQVAPA is eliminated at an early stage by the sequence purging process, the obvious solution in its case is to retain a larger number of sequences during purging. The program was modified to test this hypothesis and it was found that when the number of sequences retained was increased to 200, TSQVAPA was correctly sequenced each run by using the constants of Table 3. It was found that TSQ placed at position 130, 128, and 117 at the end of the third extension step in the three respective runs. The sequencing time doubled from 2 to 4 s.

The problem presented by HGTVVLTALGGILK is much more complex. Any approach must involve allowing sequence extension by amino acids that are not corroborated by idealized $y$ peaks. Therefore, the program was modified so that incomplete sequences were extended by all the amino acid types instead of only those supported by a peak in the idealized spectrum before repeating the tests of column three on this peptide. In all three of the tests, however, the correct sequence was still omitted from the final list. Even though the incomplete sequence, HGTVVLTAL, was extended by glycine, the correct sequence was eliminated in one of the purging steps. When the number of sequences retained during purging was increased to 500, the program still omitted the correct sequence from the final lists and processing time increased to 3 min and 45 s. When the sequencing algorithm was left unchanged and the correct sequence was manually inserted into the final lists, it placed at ranks 13, 20, and 8 when the fixed constants were used. When the scoring constants of Table 3 were used, it placed at ranks 46, 18, and 43. Thus, even if the correct partial sequence had not been eliminated in purging, the complete sequence would not have fared well in the final lists.

From examination of the mass spectrum of HGTVVLTALGGILK, it seems that the fault for the poor performance lies more with the data itself than with the sequencing algorithm. Three consecutive cleavages are very poorly represented by fragment ions in the original spectrum. As previously stated, the glycine–glycine cleavage is backed by no ions of the 11 types in the original spectrum. However, this is not the only cleavage with little supporting information in the original spectrum. The cleavage immediately preceding it (leucine–glycine) is supported only by an $a_n$ peak and the one immediately after is supported by only two $d_n$-type peaks. The presence of these ions results in two very minor peaks in the idealized spectrum. Because of the poor quality of the data for these cleavage locations, the program performance naturally diminishes as the sequences are extended into this region.

We have concluded from observation of the neural network training that performance would probably be improved significantly by training the neural networks on much larger amounts of data, derived perhaps from hundreds of spectra. For ANNs such as are used by this program to operate effectively, the data sets on which they are trained need to contain sufficiently large and varied distributions of patterns so that they represent well the problem space in general. If we assume that the patterns are well distributed over the input space, the larger the data set, the more likely that the trained network will have a stable classification power in mapping from input space to classification space. This ability is often called generalization. If the data are somewhat sparse, however, the neural networks will eventually overtrain on the training patterns and essentially memorize them without acquisition of additional ability to classify unknowns.

One way to determine if overtraining is taking place is to set aside a certain percentage of the patterns in the data set to be used only for monitoring training progress, while the rest of the patterns are used to

compute the actual weight corrections. This method was incorporated into the software used to construct and train the ANNs used in the sequencing program. In each training session, 25% of the patterns were randomly selected and set aside to be used only for monitoring training progress; the rest of the patterns were used to compute the actual weight corrections. At the end of each training epoch, the cumulative mean-squared error for both sets of patterns was written to a file. In the early epochs in training, both error measures declined, but when overtraining began to occur, the mean-squared error on the training patterns continued to decline, while the mean squared error on the monitoring patterns leveled off and actually began to increase. The left graph of Figure 9 shows the error profiles for a network like that used as network one when trained for 1000 epochs on a data set generated from all 43 reference spectra. The mean-squared error on the training patterns (lower curve) declined steadily throughout the training, but the mean-squared error of the set aside patterns (upper curve) began to increase at approximately 500 epochs. The minimum mean-squared error on the monitoring patterns was actually attained on a downward spike at epoch 140. The right side of the figure displays a similar graph for a network of type two. It too, began to overtrain within the first 500 training epochs, with minimum mean-squared error attained on the monitoring patterns at epoch 310.

We designed the neural network software used in this project to save neural networks during training when they are at their maximum generalizing potential, as estimated from monitoring the mean-squared error of the patterns set aside from computing the weight corrections. A network is saved to a disk file during training whenever its mean-squared error on the set aside patterns reaches a new minimum. For example, the software saved the two networks illustrated by Figures 9 and 10 at epochs 140 and 310, respectively. Thus, all the networks used in the testing process were saved at a fairly early stage in training. With a much larger data set, overtraining would be delayed until a much later training epoch, and network performance would increase correspondingly.

Forty-two amino acid residues in the set of reference spectra were either leucine or isoleucine. Twenty-seven of these (64.3%) were identified correctly by the differentiation routine. Only the isoleucine residue in NDIAAK was misidentified as leucine and all the rest (33.3%) were identified as indeterminate. These ambiguous residues are denoted by Xs in the final sequence list.

In the development of this program, it was observed that glutamine and lysine could not be distinguished with a high degree of confidence. To remind the user of this fact, the letter Z is used to denote both glutamine and lysine in the final list. This ambiguity may pose little difficulty if the history of the peptide is known (e.g., if it was derived from a tryptic cleavage) or if the acetylated derivative has been analyzed.

The final sequencing program was implemented in C language and X/Motif on a Silicon Graphics Indigo workstation (SGI Inc., Mountain View, CA). Figure 10 shows the program's graphical output when tested on the spectrum for VLSEG. The main window is divided into two scrollable viewports. The top displays the original preprocessed spectrum, whereas the bottom viewport depicts the idealized spectrum. When the user selects "Build sequences..." from the control menu, the program pops up a sequence dialog to display the results. If one of the displayed sequences is selected, a dotted sequence trace line will connect the tops of the peaks in the idealized spectrum.

The program also allows the user to select a peak from the original spectrum by entering a mouse click in the top viewport. Choosing "Peak info..." from the control menu displays a dialog box that contains the

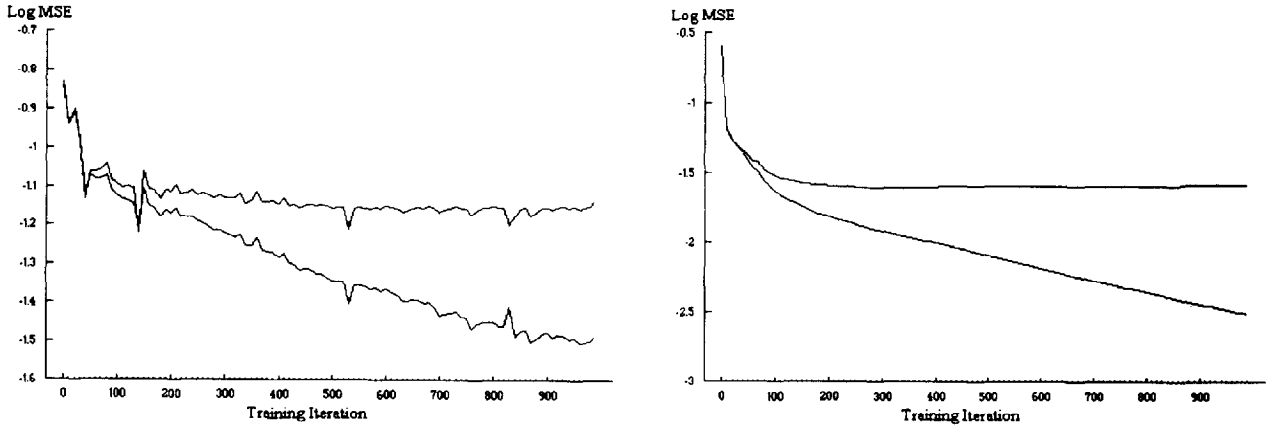



**Figure 9.** Graphs of the training progress of an ANN of type one (left) and of an ANN of type two (right). Both graphs show how the logarithm of the cumulative mean-squared error terms behaved as the networks were trained. In each graph, the bottom curve shows the mean-squared error for the patterns used to compute weight corrections, whereas the top is of the mean-squared error of the patterns set aside for monitoring training progress.

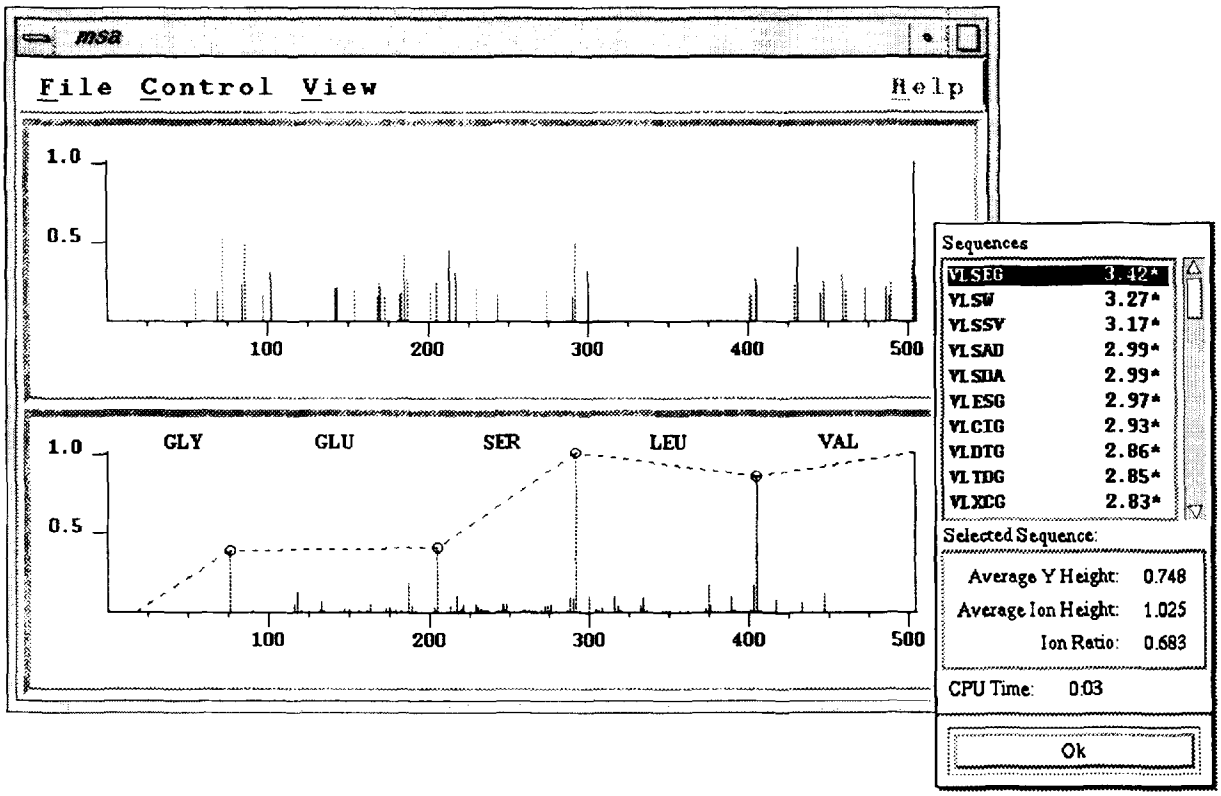



**Figure 10.** The sequencing program display as it appears in the X/Motif environment. In the actual screen display, the peaks in the upper spectrum are color coded to indicate the ion type as determined by the neural network module.

classification results of both neural networks for the chosen peak. If the user selects an idealized peak from the lower viewport, small tick marks will appear in the top viewport under the peaks from which the selected idealized peak was derived.

The sequence trace greatly assists the user in estimating the validity of the sequence interpretation. Generally for correct sequences, the trace connects the tops of prominent peaks. In some cases the user can estimate the correctness of parts of the overall sequence. In sequencing the spectrum of HGTVVLTALGGILK, for example, the program gave HGTVVLTAXZVLZ as the top rated interpretation in each of the test runs with fixed constants and in two of the test runs with separate scoring constants. When the sequence HGTVVLTAXZVLZ is selected in the sequence dialog, the segments of the trace that correspond to HGTVVLTAX connected prominent peaks, which suggests that this portion of the sequence was probably correct. The sections of the idealized spectra through which the rest of the traces passed, however, contained only small peaks.

## Conclusions

The significant time required to manually sequence peptides from high-energy CID tandem mass spectra necessitates the development of computer programs to perform this task if the speed of mass spectrometry is to be fully exploited. The results of the work described here demonstrate that artificial neural networks may be used effectively to meet this need. The program described in this paper sequences peptides in a very timely manner with the longest time taken so far being only 6 s for a peptide of nominal mass 1168 u. The success rate is not such that the user is freed entirely from scrutinizing the results. Nevertheless, the visual output provided allows the user to qualitatively judge the results so that overall interpretation time may be drastically reduced over manual interpretation.

The results obtained with this prototype algorithm are sufficiently indicative of the viability of this approach to warrant proceeding with development of neural networks trained with much larger data sets (i.e., hundreds of spectra). In addition, after the collection of a much larger set of preclassified spectra from which to generate data, we intend to investigate more thoroughly various input vector formats and the relative importance of individual elements. Such training and analysis may entail use of supercomputer capability to be accomplished in a reasonable time because the training of a single network pair on the current data sets requires several hours. We are, therefore, proceeding with plans to extend this approach to larger training sets of unmodified peptide spectra as well as to spectra of modified peptides.

Intuitively, it seems that this algorithm also could be applied to the interpretation of low-energy CID mass spectra of peptides; however, because of the different character of low-energy CID spectra, such an adaptation of the program would require training of the neural networks on such spectra and the rules governing the presence of fragmentation ions also would have to be modified. The current algorithm assumes the mass resolution of the peak data to be sufficiently high that peak masses may be converted to integers and that subsequent mass comparisons between the peaks may be safely compared to the nominal amino acid residue masses with no margin of error. Because low-energy CID spectra are typically recorded at a much lower mass resolution than high-energy CID, a margin of error might have to be introduced to compare mass differences between ions. This would undoubtedly introduce additional programming complexity.

## Acknowledgments

We would like to thank Wade M. Hines for the spectrum of the peptide DVVLVDAGLK. Work was supported in part by NIH grant EY08239. The JEOL HX110/HX110 instrument was funded in part by NSF grant DIR 88-04502.

## References

1. Biemann, K. *Annu. Rev. Biochem.* **1992,** *61,* 997–1010.
2. Ishikawa, K.; Niwa, Y. *Biomed. Environ. Mass Spectrom.* **1986,** *13,* 373–380.
3. Hamm, C. W.; Wilson, W. E.; Harvan, D. J. *Comput. Appl. Biosci.* **1986,** *2,* 115–118.
4. Johnson, R. S.; Biemann, K. *Biomed. Environ. Mass Spectrom.* **1989,** *18,* 945–957.
5. Hines, W. M.; Falick, A. M.; Burlingame, A. L.; Gibson, B. W. *J. Am. Soc. Mass Spectrom.* **1992,** *3,* 326–336.
6. Biemann, K. *Meth. Enzymol.* **1990,** *193,* 886–887.
7. Johnson, R. S.; Martin, S. A.; Biemann, K. *Int. J. Mass Spectrom. Ion Processes* **1988,** *86,* 137–154.
8. Johnson, R. S.; Martin, S. A.; Biemann, K.; Stults, J. T.; Watson, J. T. *Anal. Chem.* **1987,** *59,* 2621–2625.
9. Johnson, R. S. Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1988.
10. Downard, K. M.; Biemann, K. *J. Am. Soc. Mass Spectrom.* **1993,** *4,* 874–881.
11. Mueller, P.; Lazzaro, J. In *Neural Networks for Computing;* Denker, J., Ed.; American Institute of Physics: New York, 1986.
12. Scarberry, R. E.; Zhang, Z. *Proceedings of the Artificial Neural Networks in Engineering Conference;* St. Louis, MO, 1991; p 351.
13. Martin, G. L.; Pittman, J. A. *Proceedings of the IEEE Conference on Neural Information Processing Systems;* San Mateo, CA, November 1990; pp 405–414.
14. Goodacre, R.; Karim, A.; Kaderbhai, M. A.; Kell, D. B. *J. Biotechnol.* **1994,** *34,* 185–193.
15. Lippman, R. P. *IEEE ASSP Mag.* **1987,** *4,* 4–22.
16. Kolmogorov, A. N. *Dokl. Akad. Nauk USSR* **1957,** *114,* 953–956.
17. Hecht-Nielsen, R. *Proceedings of the International Conference on Neural Networks II;* IEEE Press: New York, 1987; pp 131–139.
18. Werbos, P. J. Doctoral Dissertation, Appl. Math., Harvard University, 1974.
19. Rumelhart, D. E.; McClelland, J. L., Eds. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, I, & II;* MIT Press: Cambridge, MA, 1986.
20. Jurs, P. C. In *Computer Software Applications in Chemistry;* Wiley: New York, 1986; p 186.